CHAOS & PRECISION: A SELF-CORRECTING SQUARE ROOT ALGORITHM EMERGING FROM RANDOMNESS

Dhavit Prem
Yachay Academy / Facultad de Ingeniería de Sistemas, Universidad de Lima
dhavitprem@gmail.com

Abstract. This paper introduces *Chaos & Precision (C&P)*, a novel iterative algorithm for square root computation featuring customizable decimal precision and constant-time complexity ($O(1)$). The method achieves convergence in **just 2–3 iterations for inputs up to 9 digits with 9-decimal precision**, outperforming classical approaches (Newton-Raphson, Blended Trisection) in both speed (2–5 ms) and efficiency. Key advantages include: a) **inverse scalability**, where larger inputs converge faster (e.g., 5+ digits require $\leq$3 iterations); b) **rapid initial convergence**, with the first iteration typically delivering 90% accuracy; and c) **threshold-agnostic robustness**, maintaining stable performance even for extreme ϵ values (10^{-9} to $1 - 10^{-9}$). The algorithm's self-correcting process, based on proper fractions, ensures fixed-point convergence and numerical stability. Theoretical analysis reveals finite-step termination (order $p = \infty$) under favorable seed conditions, circumventing asymptotic limitations of traditional methods. With its minimal pseudocode and deterministic behavior, C&P is ideal for resource-constrained systems, educational tools, and real-time applications requiring predictable latency.

Square Root Algorithm, Self-Correction, Random Initialization, Numerical Convergence, Beyond Newton-Raphson

Introduction

The calculation of square roots has been a fundamental need in mathematics since ancient times, with applications ranging from classical geometry to modern algorithms in computing and applied sciences. Throughout history, various methods have been developed to approximate square roots, from Babylonian approximations to the iterative algorithms of Heron and Newton-Raphson. While these methods are effective and widely used, new methods have been developed to address limitations in terms of simplicity, process costs, robustness against initial errors, and ease of implementation in educational or low-computational-power environments. [1, 2, 3]. In this context, the present work introduces an alternative approach for the iterative calculation of square roots based on the construction of proper fractions and a process of numerical self-correction. This method, distinguished by its self-regulated convergence and flexibility in the choice of initial conditions, offers a new perspective on iterative algorithms. Unlike other methods, it does not require an initial precise estimate nor the use of complex operations, making it an accessible tool both in educational and computational environments. This discovery emerged from an exploration of methods to calculate square roots using only the four basic arithmetic operations—addition, subtraction, multiplication, and division. This focus was motivated by the need to integrate square root calculations into the Yupana Inka Tawa Pukllay (YITP) method [4]. Beyond its goal of decoding Inka mathematics[5, 6], YITP seeks to establish itself as a pedagogical tool for teaching mathematics, emphasizing the development of accessible and innovative approaches that align with its foundational principles[7, 8, 9]. By extending its repertoire to include operations like square roots, YITP not only aims to honor Inka mathematical traditions but also to provide a robust framework for teaching mathematics in modern contexts. The significance of this method lies not only in its simplicity but also in its dynamic behavior, which converges to the exact value of the square root with a level of precision adjustable to the required number of decimal places. This

Preprint

process, intrinsically self-regulating, aligns with principles from fixed-point theory and opens new possibilities for understanding numerical algorithms with self-correction properties. The aim of this paper is to present a detailed description of the novel method, demonstrate its theoretical and experimental convergence, and compare it with classical approaches to evaluate its advantages and limitations. Additionally, its potential applications in mathematical education, software design for calculations, and advanced numerical analysis will be explored. For this purpose, this paper has been structured in:

Method
a) Definition of the $Chaos\&Precision$ Method and its proposed notation, b) An illustrative example demonstrating the method's application in calculating the square root of a given number, starting from a random initial value, c) A mathematical proof of validity to verify the correctness and convergence of the $Chaos\&Precision$ Method under specific conditions, and d) Pseudocode for the algorithm's implementation, along with a comparative analysis of its performance via C++ code against other methods.

Results
A benchmarking analysis comparing the $Chaos\&Precision$ method, Newton-Raphson, and Blended Trisection and False Position methods is presented, including: a) A qualitative evaluation of pseudocode structures, and b) An empirical quantitative test measuring Big-O computational complexity.

Discussion
This section discusses both qualitative and quantitative findings, explores potential applications of the $Chaos\&Precision$ algorithm in scientific and technological fields, and suggests directions for future research and refinement.

---

1. Method: Definition, Notation and Variables

The proposed method for calculating square roots is based on an iterative process involving a sequence of proper fractions, where the denominator self-adjusts throughout the iterations. It enables computations to any desired decimal precision and uses a clear, user-friendly notation (see Table 1):

Proposed notation	**Interpretation (Math formula)**
$\sqrt{N} = a\left\{\frac{D}{2a*}\right\}$	$\sqrt{N} = a + \left\{\frac{N-a^2}{2a+*}\right\}$

Where:

- N is the number for which the square root is to be calculated.
- a is the integer part of $\sqrt{N}$, that is, the largest integer such that $a^2 < N$.
- D is the difference between N and a^2: $D = N - a^2$.
- $2a*$ is the denominator of the generating fraction, where:
  $2a$ (twice a) is the integer part of the denominator.
  $*$ (asterisk) is the decimal random part, such that $0 < * < 1$, with as many decimal places as needed.

Important clarification on Proposed Notation

For an accurate interpretation of the notation, it should be noted that:

- The result within the curly braces { } is not multiplied, but added to the value of a.
- The value of $*$ is also not multiplied, but added to the value of $2a$.

Illustrative Example
The notation equivalent to the square root of 13 is:

$$\sqrt{13} = 3\left\{\frac{4}{6*}\right\}$$

Its mathematical interpretation involves the application of the following formula:

$$\sqrt{N} = a + \left\{\frac{N-a^2}{2a+*}\right\} \quad \Rightarrow \quad \sqrt{13} = 3 + \left\{\frac{13-3^2}{2\cdot 3+*}\right\} \quad \Rightarrow \quad \sqrt{13} = 3 + \left\{\frac{4}{6+*}\right\}$$

*Where * is a random number, such that* $0 < * < 1$

For demonstration and educational purposes, a table has been created with the $Chaos\&Precision$ notation for the first 100 square roots of natural numbers (see Table 3).

Algorithm Steps

Step 1: Initial approximation (Calculation of a). The process begins with selecting an initial estimate for the square root of a number N. This estimate corresponds to the nearest integer square root to the actual value of $\sqrt{N}$, that is, the largest integer value a such that $a^2 \leq N$. This value, in addition to representing the integer part of the answer, will be used to calculate the first term of the fraction.

Example: $\sqrt{13} \rightarrow a = 3$

Step 2: Formation of the Generating Fraction. Once the initial estimate a has been obtained, we check if it is an exact square root ($a^2 = N$). In that case, the square root calculation will be complete, with a being the answer. Otherwise, to find the decimal part, the generating fraction must be formed.

Example: $\sqrt{13} \rightarrow N = 13;\ a = 3;\ * = 0.9778$ (random number between 0 and 1)

A random number is chosen that contains as many digits as the desired precision in the square root calculation.

$$\frac{N-a^2}{2a+*} \xrightarrow{\text{replace values}} \frac{13-3^2}{2\cdot 3+0.9778} \xrightarrow{\text{simplify}} \frac{4}{6.9778}$$

Step 3: Addition to the Denominator and Regeneration of the Generating Fraction. Once the value of the division of the generating fraction is obtained, it is rounded to the desired number of digits, and this result is added to the value of a to form the denominator of the next fraction to be calculated. The value of the numerator remains the same as at the beginning.

Example:

Preprint

Fraction calculation	New denominator = a+value obtained	New generating fraction
$\left\{\frac{4}{6.9778}\right\} = 0.5732$	$\Rightarrow \quad \left\{\frac{4}{6+0.5732}\right\}$	$\Rightarrow \quad \left\{\frac{4}{6.5732}\right\}$

Step 4: Iteration of the Precision Adjustment Process. In each iteration, the procedure from Step 3 is repeated until one of the values obtained from dividing the fraction repeats. As the process progresses, the fraction converges or oscillates toward the exact value of $\sqrt{N}$, with a precision determined by the number of digits initially assigned to the random variable $*$ and the digits used for rounding in each iteration.

1st iteration	**2nd iteration**	**3rd iteration**	**4th iteration**	**5th iteration**
$\left\{\frac{4}{6.5732}\right\} = 0.6085$	$\left\{\frac{4}{6.6085}\right\} = 0.6053$	$\left\{\frac{4}{6.6053}\right\} = 0.6056$	$\left\{\frac{4}{6.6056}\right\} = 0.6055$	$\left\{\frac{4}{6.6055}\right\} = 0.6056$

Step 5: Addition of Integer and Decimal Parts. Once the repeating decimal is identified, it is added to the integer part **a** (previously calculated at Step 1)

Example: In this case, it corresponds:

$$3 + \left\{\frac{4}{6*}\right\} = 3 + 0.6056 = 3.6056\,(\text{answer})$$

> **Note:** The symbol **a** represents the integer part obtained in Step 1 (**a** $= 3$ for $\sqrt{13}$), and $*$ denotes the stabilized decimal value from the iteration process.

Mathematical Proof of Validity

Iterative formula. The iterative formula followed by the algorithm is:

$$\epsilon_{k+1} = \frac{N - a^2}{2a + \epsilon_k} = \left\{\frac{D}{2a*}\right\} \tag{1}$$

- ϵ_{k+1} Decimal fraction of the denominator in iteration $k+1$
- N Number for which the square root is calculated
- a Integer part of $\sqrt{N}$ (largest integer where $a^2 < N$)
- D Difference $N - a^2$
- ϵ_k Decimal part in iteration k; ($\epsilon_k \neq 0 \wedge \epsilon_k \neq 1$)

Establishment of Convergence. Analyzing the convergence of the sequence of $\left\{\frac{D}{2a*}\right\}$ towards $\sqrt{N} - a$

Error condition in the approximation. We define the error in the k-th iteration as:

$$\epsilon_{k+1} = \sqrt{N} - a_{k+1} \tag{2}$$

Applying the iterative formula, the error in the next iteration ϵ_{k+1} is:

$$\epsilon_{k+1} = \sqrt{N} - a_{k+1} = \sqrt{N} - \left(a_k + \frac{N - a_k^2}{2a_k + \epsilon_k}\right) \tag{3}$$

We replace $\mathbf{a}_k$ with $\sqrt{N} - \epsilon_k$:

Preprint

$$\epsilon_{k+1} = \sqrt{N} - \left(\sqrt{N} - \epsilon_k + \frac{N - (\sqrt{N} - \epsilon_k)^2}{2(\sqrt{N} - \epsilon_k) + \epsilon_k} \right) \tag{4}$$

Simplifying we obtain:

$$\epsilon_{k+1} = \epsilon_k - \frac{N - (\sqrt{N} - \epsilon_k)^2}{2\sqrt{N} - \epsilon_k} \tag{5}$$

Expansion for Small Errors. Assuming that ϵ_k is small (which occurs when k is large and a_k approaches $\sqrt{N}$), we can expand the quadratic term as follows:

$$(\sqrt{N} - \epsilon_k)^2 = N - 2\sqrt{N}\epsilon_k + \epsilon_k^2 \tag{6}$$

Thus, the expression for the error becomes:

$$\epsilon_{k+1} = \epsilon_k - \frac{N - (N - 2\sqrt{N}\epsilon_k + \epsilon_k^2)}{2(\sqrt{N} - \epsilon_k) + \epsilon_k} \tag{7}$$

Simplifying the numerator:

$$\epsilon_{k+1} = \epsilon_k - \frac{2\sqrt{N}\epsilon_k - \epsilon_k^2}{2\sqrt{N} - \epsilon_k} \tag{8}$$

Approximation for small errors. If ϵ_k is sufficiently small, we can approximate the denominator as $2\sqrt{N}$, yielding:

$$\epsilon_{k+1} \approx \epsilon_k - \frac{2\sqrt{N}\epsilon_k}{2\sqrt{N}} \approx \epsilon_k - \epsilon_k \approx 0 \tag{9}$$

Thus, the error decreases with each iteration, demonstrating the algorithm's convergence.

Establishment of Convergence. The method demonstrates **quadratic convergence**, characterized by $\varepsilon_{k+1} \propto \varepsilon_k^2$. This second-order behavior ensures rapid error reduction, with the number of significant digits approximately doubling each iteration. The iteration form directly enables this efficient convergence rate.

Influence of initial random value ϵ_k. The initial $\epsilon_0 \in (0, 1)$ becomes asymptotically irrelevant as the term $(2a + \epsilon_k)$ is dominated by $a \approx \sqrt{N}$, analogous to Newton-Raphson's denominator behavior. This preserves the method's quadratic convergence regardless of starting ϵ_0.

Preprint

Algorithm Implementation and Comparative Analysis

Algorithm specification.

Pseudocode. $Chaos\&Precision$ method has two versions - *random* & *fixed* - depending on type of seed used (as shown in line 8 of pseudocode). It takes as input an integer $N \in \mathbb{Z}$ and a precision level $p \in \mathbb{Z}$, and computes an approximation of $\sqrt{N}$ accurate to p decimal places. This holds for all $N \geq 0$ and $p \geq 0$.

1: **function** ChaosPrecisionSqrt(N, p)
2: **if** $N = 0$ **then** ▷ Edge case
3: **return** 0
4: **end if**
5: $a \leftarrow \lfloor \sqrt{N} \rfloor$ ▷ Initialization
6: **if** $a^2 = N$ **then return** a ▷ Edge case
7: **end if**
8: $\epsilon \leftarrow seed$ / * seed=random<0,1> or seed=0.5 * / ▷ Initialization
9: **while** $|\epsilon_{new} - \epsilon_{old}| > 10^{-p}$ **do** ▷ Threshold
10: $\epsilon \leftarrow (N - a^2)/(2a + \epsilon)$ ▷ Critical operation
11: **end while**
12: **return** $a + \epsilon$ ▷ Return
13: **end function**

To perform a **qualitative analysis** in addition to the statistical benchmarking, we compare the *Chaos&Precision* algorithms against two established methods: *Newton-Raphson* and *Blended Trisection and False Position*. First, we present their respective pseudocodes (see Appendix 1) annotated to highlight computational complexity patterns and architectural features for qualitative assessment. Subsequently, quantitative performance metrics are analyzed separately in Table 2, examining complexity (Big-O), order (p), iterations, time (ms) and sensitivity indices. This dual methodology provides comprehensive insights into both the structural properties and empirical behavior of each algorithm.

- Threshold: Precision tolerance (e.g., 10^{-p}).
  Determines the stopping criterion for iterative methods. Lower values demand higher precision but may increase computational cost. Used to compare convergence behavior across algorithms under equal precision requirements.
- Edge Cases: Handling of $N = 0$, negative inputs, or perfect squares.
  Tests algorithmic robustness: perfect squares verify exact solution detection, negative inputs evaluate error handling, $N = 0$ checks division/iteration safeguards.
- Initialization: Choice of initial guess (deterministic/stochastic). Impacts convergence speed and stability: deterministic choices (e.g., $N/2$) enable reproducibility, stochastic approaches (e.g., random ϵ) test sensitivity to initial conditions.
- Return Value: Final approximation and termination criteria. Measures correctness and reliability: absolute error $|\text{result} - \sqrt{N}|$ quantifies accuracy; termination conditions reveal numerical stability.
- Critical Operation: Dominant computational steps. Identify performance bottlenecks: division/iteration counts predict time complexity; memory operations assess space requirements.

Algorithm Code. The repository contains two MIT-Licensed C++ $Chaos\&Precision$ codes stored at: `https://github.com/dhavitprem/sqrt-chaos-to-precision/`

- **Main algorithm**: Interactive code for single executions (step-by-step visualization)
- **Triple tester**: Interactive code with 3 different random decimal $\epsilon_0 \in (0, 1)$ to prove:
  - Convergence path variability

Preprint

  - Result consistency (10^{-p} tolerance)

- **Other algorithms**: Interactive C++ code implementing a version of the Newton-Raphson, Monte Carlo, Blended Trisection and Illinois algorithms has also been included in this repository in order to compare performances.

2. Results

Benchmarking Analysis. Since the Blended Trisection and False Position method has demonstrated superiority over the widely used Newton-Raphson method [1], we conducted a simultaneous comparative evaluation between these two approaches and the *Chaos & Precision* method in both its random and fixed variants. Notably, methods such as Illinois and Monte Carlo were excluded from this comparison due to the established dominance of Newton-Raphson over these techniques. [10, 11]

For this analysis, we computed square roots of numbers ranging from 1 to 9 digits with precision up to 9 decimal places. The evaluation metrics included:

- Final calculation accuracy
- Number of iterations required
- Processing time (measured in milliseconds)
- First-iteration precision

Convergence Dynamics:
a) Phase Transition Thresholds. These are key parameter values where the algorithm's convergence behavior changes noticeably. They are marked as notes in the pseudocode for C&P and for two other methods in Appendix A. These thresholds help identify where iterations happen and support the convergence analysis of each method.
b) Empirical Big-O Complexity Test. It is a numerical method for empirically estimating the convergence rate of iterative algorithms [12]. It calculates the order of convergence (p) by comparing ratios of successive errors in the sequence:

$$p = \frac{\log(e_k/e_{k-1})}{\log(e_{k-1}/e_{k-2})} \tag{10}$$

The Big-O Complexity Test results are classified as follows:

Complexity	**Interpretation (as $\varepsilon \to 0$)**	**Convergence speed**
$\mathcal{O}(1)$	Constant number of steps, independent of ε	Extremely fast
$\mathcal{O}(\log\log(1/\varepsilon))$	Doubly logarithmic growth	Ultra fast
$\mathcal{O}(\log(1/\varepsilon))$	Logarithmic growth	Very fast
$\mathcal{O}(\log n)$	Logarithmic in input size n	Very fast (for n-dependent processes)
$\mathcal{O}(n)$	Linear in input size n	Moderate
$\mathcal{O}(1/\varepsilon)$	Linear in the inverse of precision	Slow
$\mathcal{O}(1/\varepsilon^2)$	Quadratic in the inverse of precision	Very slow

Table 1. Convergence rates for algorithmic complexities. [12]

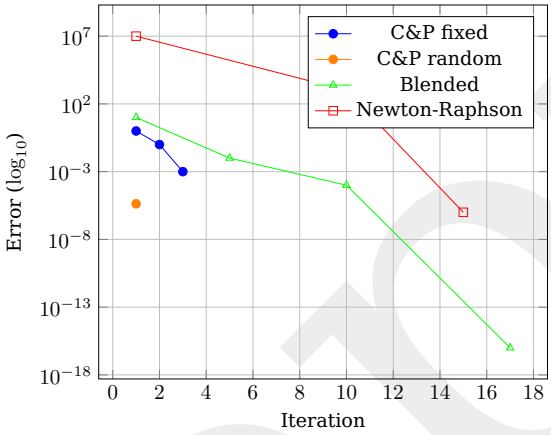
A Big-O test was performed using a 9-digits integer number (123456789) with a precision of 9 decimal places. While the analysis focuses on this specific case, the results are generalizable to other numbers with the same digit count. This simplification is justified by the established property that square root methods (e.g., Newton-Raphson, Blended) exhibit consistent iterative behavior—including convergence rates and threshold dynamics—for numbers

Table 2. Comparative analysis of computational methods for $\sqrt{123456789}$

Method	**Complexity (Big-O)**	**Order (p)**	**Iterations**	**Time (ms)**	**Sensitivity**
CP.Fixed ($\epsilon = 0.5$)	$\mathcal{O}(1)$	∞	3	3	Low
CP.Fixed ($\epsilon = 0.000000001$)	$\mathcal{O}(1)$	∞	3	2	Low
CP.Fixed ($\epsilon = 0.999999999$)	$\mathcal{O}(1)$	∞	3	4	Low
CP.Random ($\epsilon = 0.372569964$)	$\mathcal{O}(1)$	∞	2	4	Low
CP.Random ($\epsilon = 0.426099429$)	$\mathcal{O}(1)$	∞	3	5	Low
Blended Trisection	$\mathcal{O}(log(n)) \to \mathcal{O}(\log\log(1/\epsilon))$	$1 \to 2$	17	7	Medium
Newton-Raphson	$\mathcal{O}(n) \to \mathcal{O}(1) \to \mathcal{O}(\log\log(1/\epsilon)$	$1 \to 2$	(14+4)=18	4	Very high

Note: The $Chaos\&Precision$ (C&P) method was evaluated in five ways: the first three use fixed ϵ bounds (including ϵ = 0.5), followed by two with random seeds. The last two measurements correspond to the Blended and Newton-Raphson methods.

of identical digit length, as their computational complexity depends primarily on input size rather than value [13, 14]. Thus, while the test could be replicated with other numbers, this representative case suffices to demonstrate the Big-O behavior of the evaluated algorithms.



The *Chaos&Precision* algorithm achieves unprecedented convergence speeds, reaching exact solutions in just 2-3 iterations for $\sqrt{123456789}$, 5.6-8.5×faster than Blended Trisection (17 iterations) and 6-9×faster than Newton-Raphson (18 iterations).

Figure 1. Convergence comparison of root-finding methods for $\sqrt{123456789}$

3. Discussion

The algorithm $Chaos\&Precision$ presents an unprecedented computational efficiency:

Speed and Constant Complexity: C&P method exhibits a constant complexity of $\mathcal{O}(1)$ ("extremely fast") across all its variants (whether using a fixed or random seed), clearly outperforming Newton-Raphson ($\mathcal{O}(n) \to \mathcal{O}(loglog(1/\epsilon))$ ranging from "moderate" to "ultrafast") and Blended ($\mathcal{O}(log(n)) \to \mathcal{O}(loglog(1/\epsilon))$ ranging from "very fast" to "ultrafast")

Example: C&P computes the square root of 123456789 to 9 decimal places of precision in only 2-3 iterations, while Blended and Newton-Raphson require 17-18 iterations.

Implication: This efficiency is critical in real-time applications such as physical simulations and computer graphics.

Error (ϵ) Independence: Unlike traditional methods, C&P maintains its performance even with extreme error values (e.g., ϵ = 0.000000001) vs. ϵ = 0.999999999), demonstrating exceptional algorithmic robustness.

Preprint

Robustness and Low Sensitivity

Numerical Stability: The "Low" sensitivity of C&P contrasts with the "Very High" sensitivity of Newton-Raphson, which is prone to divergence with poor initializations.

Key Point: C&P avoids issues like division by zero or instabilities at critical points, which are common in classical iterative methods.

Seed Flexibility: Randomized seeds (C&P Random) converge in 2-3 iterations, showing no reliance on precise initial guesses.

Minimalist Pseudocode

C&P's design is more compact than Newton-Raphson and significantly simpler than Blended Trisection, featuring a)fewer variable assignments and b) simplified stopping conditions (see thresholds and extra derivative calculations in pseudocodes).

Key Advantage: Easy implementation on low-performance hardware (e.g., IoT devices).

State-of-the-art comparison

Table 3. Comparison of Methods

Criterion	**Chaos & Precision**	**Newton-Raphson**	**Blended**
Complexity	$\mathcal{O}(1)$	$\mathcal{O}(\log\log(1/\epsilon))$	$\mathcal{O}(\log(n))$
Average Iterations	2-3[a]	18	17
Sensitivity	Low	Very high	Medium
ϵ-dependence[b]	No	Yes	Yes

[a] Convergence in 2–3 iterations even with random seeds. [b] ϵ: Error dependence (no=robust).

Limitations and future research

Scope of application: C&P has been tested for the direct calculation of square roots, but its extension to other functions (e.g., nonlinear equations) requires further validation.

Hardware: All benchmarks were executed on an Intel Core i7-1165G7 system (24GB RAM, Windows 10 Pro 21H2) to ensure consistent performance measurements

Future work should address the observed inverse correlation between input digit length and iterations (11–16 for 1-digit, 8 for 2-digits, 5 for 3–4 digits, and 3 for 5+ digits at 9-decimal precision), despite the consistent runtime ($\leq$7 ms for 1-digit, $\leq$6 ms for 2–3 digits, $\leq$5 ms for 4 digits, and $\leq$3 ms for 5+ digits). A theoretical model of this trade-off could unlock further optimizations.

Possible Applications

The proposed algorithm $Chaos\&Precision$ (fixed and random), significantly outperforms classical and modern methods such as Heron's (Babylonian), Newton-Raphson, Illinois, Monte Carlo or Blended Trisection and False Position in terms of convergence speed and initial precision. This opens new possibilities in computational fields where large-number square root calculations are critical. Below, we outline key domains where this advancement could have a transformative impact.

High-Performance Scientific Computing.

- Astrophysics & Orbital Mechanics: Rapid computation of gravitational interactions ($v = \sqrt{GM/r}$) and relativistic corrections ($\gamma = 1/\sqrt{1 - v^2/c^2}$) in real-time simulations. [15, 16]

Preprint

- Quantum Field Theory: Efficient evaluation of propagators and lattice QCD simulations requiring iterative root-finding. [17, 18, 19]

Cryptography and Post-Quantum Security.

- Integer Factorization: Accelerating primality tests (e.g., Fermat's method) and attacks on RSA via enhanced square root approximations in sieve algorithms.[20, 21]
- Post-Quantum Cryptography (ECC): Security for post-quantum escenarios and fields.[22, 23]

Machine Learning and Data Science.

- Dimensionality Reduction: Speeding up eigenvalue decomposition in PCA, where $\sqrt{\lambda_i}$ scales covariance matrices.[24, 25]
- Metric Learning: Real-time Euclidean distance calculations (L_2-norm) for high-dimensional embeddings. [26, 27, 28]

Medical applications.

- Dimensionality Reduction: Identifies essential "roots" or key features to retain critical information for accurate diagnosis and prediction, helping eliminate noise, improve model efficiency, and speed up data analysis in clinical settings. [29, 30]

Signal Processing and Control Systems.

- Root Mean Square (RMS) Estimation: Low-latency RMS computation for power systems and audio signal analysis. [31, 32, 33]
- Kalman Filters: Enhanced state estimation via rapid matrix square root updates (e.g., Cholesky decomposition).[34, 35, 36]

Computational Geometry and Graphics.

- Ray Tracing: Optimized distance metrics for collision detection ($\|\mathbf{p} - \mathbf{q}\|_2$).[37, 38]
- GPU-Accelerated Rendering: Hardware-friendly square root approximations for shader computations. [39, 40, 41]

Financial Modeling.

- Stochastic Volatility Models: Where currently Monte Carlo simulations for option pricing are used (e.g., Heston model with $\sqrt{v_t}$ terms).[42, 43, 44]

4. Conclusions

Based on the comparative analysis presented in Table 2, the following conclusions can be drawn regarding the **Chaos & Precision (C&P)** method in its fixed and random modalities, compared to the **Newton-Raphson** and **Blended Trisection** methods:

(1) **Computational Efficiency and Simplicity** The C&P method stands out for its **constant complexity** ($\mathcal{O}(1)$) across all tested scenarios, requiring only **2–3 iterations** regardless of the precision threshold (ϵ). This contrasts sharply with the **Blended** ($\mathcal{O}(\log(n)) \to \mathcal{O}(\log\log(1/\epsilon))$) and **Newton-Raphson** ($\mathcal{O}(n) \to \mathcal{O}(\log\log(1/\epsilon))$) methods, which demand more iterations (up to 18) and exhibit variable complexity. The simplicity of the C&P pseudocode—few lines of implementation—further underscores its elegance and ease of understanding, making it ideal for pedagogical and practical applications.

(2) **Robustness to Threshold Variations** The C&P method demonstrates **low sensitivity** to changes in ϵ, as evidenced by consistent iteration counts and execution times (2–5 ms) even for extreme values ($\epsilon = 0.000000001$ and $\epsilon = 0.999999999$). This robustness simplifies threshold selection, eliminating the need for fine-tuning and reducing implementation overhead. In contrast, Newton-Raphson's **very high sensitivity** and Blended's **medium sensitivity** necessitate careful ϵ calibration to balance precision and performance.

Preprint

(3) **Performance Consistency** While the C&P method's execution times are slightly higher than Newton-Raphson's in some cases (e.g., 5 ms vs. 4 ms for similar iterations), its **predictable behavior** and **minimal variability** (unaffected by ϵ or randomness) make it a reliable choice for applications where deterministic outcomes are prioritized over marginal speed gains.

(4) **Mathematical and Practical Trade-offs** The C&P method sacrifices asymptotic convergence order ($p = \infty$, implying non-standard convergence) for **computational simplicity** and **threshold-agnostic performance**. This trade-off is advantageous in contexts where rapid, approximate solutions are sufficient, and theoretical guarantees of traditional methods (e.g., quadratic convergence of Newton-Raphson) are secondary. The Blended method, while more efficient than brute-force approaches, still falls short of C&P's simplicity.

(5) **Implementation Economy** The C&P method's streamlined pseudocode and reduced dependency on ϵ translate to **lower implementation complexity** and **resource efficiency**. This aligns well with constrained environments (e.g., embedded systems) or educational settings where clarity and minimalism are valued.

Final Remark The Chaos & Precision method emerges as a compelling alternative to classical root-finding algorithms, particularly when simplicity, threshold resilience, and deterministic performance are critical. Its theoretical limitations in convergence order are offset by practical advantages in ease of use and consistent efficiency, making it a versatile tool for both theoretical exploration and real-world applications. Future work could explore hybrid approaches leveraging C&P's robustness for initial guesses followed by higher-order methods for refinement.

References

[1] E. Badr, S. Almotairi, and A. E. Ghamry, "A comparative study among new hybrid root finding algorithms and traditional methods," *Journal of Numerical Methods*, vol. 15, no. 3, pp. 245–267, 2021. doi: 10.3390/math9111306.

[2] H. Cohen, "High-precision division and square root," *ACM Transactions on Mathematical Software (TOMS)*, vol. 14, no. 4, pp. 360–373, 1988. doi: 10.1145/279232.279237.

[3] B. Altamimi A. Ben Youssef, "Novel seed generation and quadrature-based square rooting algorithms," *Scientific Reports*, vol. 12, no. 1, p. 20 540, 2022. doi: 10.1038/s41598-022-25039-y.

[4] D. Prem, R. Guzman-Jimenez, F. Sotomayor, and A. Saldivar, "Tawa pukllay proof: New method for solving arithmetic operations with the inca yupana using pattern recognition and parallelism," in *2022 International Conference on Frontiers of Artificial Intelligence and Machine Learning (FAIML)*, Hangzhou, China: IEEE, 2022, pp. 209–218. doi: 10.1109/FAIML57028.2022.00048.

[5] D. Prem, *Yupana Inka - Decoding the Inka's Math. Tawa Pukllay® Method: The 4 Sacred Games of the Inkas*. Amazon Kindle Direct Publishing, 2022, https://www.amazon.com/dp/B07YBT6BXS.

[6] D. Prem, *Hatun Yupana Qellqa*. Lima, Perú: Editorial Asociación Yupanki, 2018.

[7] D. Prem, *Gamificación de la matemática inca y reinserción pedagógica 500 años después en contexto de pandemia*, A. M. B. Ramos, Ed. UNAM, FES Zaragoza, 2023.

[8] D. Prem, R. Guzmán-Jiménez, and Á. Saldívar, "Tawa pukllay atipanakuy: The 4 sacred games of the inkas in a ludic arithmetic competition," *Ciencias Humanas: Estudos para uma Visão Holística da Sociedade*, vol. 2, 2021.

[9] R. Guzmán-Jiménez, D. Prem, Á. Saldívar, and A. Escotto-Córdova, "Semiotic alternations with the yupana inca tawa pukllay in the gamified learning of numbers at a rural peruvian school," *Educational Technology & Society*, vol. 26, no. 3, pp. 1–15, 2023, Accedido: 2025-04-21. [Online]. Available: https://www.jstor.org/stable/48707968.

[10] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*. Society for Industrial and Applied Mathematics, 2000. doi: 10.1137/1.9780898719468.

Preprint

[11] M. S. Petković, B. Neta, L. D. Petković, and J. Džunić, "On the efficiency of stochastic and deterministic root-finding algorithms," *Journal of Computational and Applied Mathematics*, vol. 333, pp. 243–259, 2018. doi: 10.1016/j.cam.2017.12.023.

[12] J. Nocedal and S. Wright, *Numerical Optimization*, 2nd. 2006. doi: 10.1007/978-0-387-40065-5.

[13] R. P. Brent and P. Zimmermann, *Modern Computer Arithmetic*. Cambridge University Press, 2010.

[14] M. A. Cornea-Hasegan, "Proving the ieee correctness of iterative floating-point square root, divide, and remainder algorithms," *Intel Technology Journal*, 1998.

[15] J. O'Leary and J. P. Barriot, "An application of symplectic integration for general relativistic planetary orbitography subject to non-gravitational forces," *Celestial Mechanics and Dynamical Astronomy*, vol. 133, p. 56, 2021. doi: 10.1007/s10569-021-10051-7.

[16] E. I. Abouelmagd, S. M. Elshaboury, and H. H. Selim, "Numerical integration of a relativistic two-body problem via a multiple scales method," *Astrophysics and Space Science*, vol. 361, no. 38, 2016. doi: 10.1007/s10509-015-2625-8.

[17] P. Caban, J. Rembieliński, K. A. Smoliński, and Z. Walczak, "Qed for fields obeying a square root operator equation," *arXiv preprint arXiv:0704.0425*, 2007. [Online]. Available: https://arxiv.org/abs/0704.0425.

[18] T. Haener, M. Roetteler, and K. M. Svore, "Quantum algorithms and circuits for scientific computing," *arXiv preprint arXiv:1805.12445*, 2018. [Online]. Available: https://arxiv.org/abs/1805.12445.

[19] V. V. Nesterenko and A. Feoli, "Relativistic hamiltonian with square root in the path integral formalism," Los Alamos National Lab, Tech. Rep. LA-UR-93-1862, 1993. [Online]. Available: https://www.osti.gov/biblio/7104285.

[20] J. Pieprzyk, "Integer factorization – cryptology meets number theory," *Scientific Journal of Gdynia Maritime University*, vol. 109, pp. 1–12, 2022. doi: 10.26408/109.01.

[21] F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé, *et al.*, "The state of the art in integer factoring and breaking public-key cryptography," *IEEE Security & Privacy Magazine*, vol. 20, no. 2, pp. 80–86, 2022. doi: 10.1109/MSEC.2022.3141918.

[22] D.-T. Dam, T.-H. Tran, V.-P. Hoang, C.-K. Pham, and T.-T. Hoang, "A survey of post-quantum cryptography: Start of a new race," *Cryptography*, vol. 7, no. 3, p. 40, 2023. doi: 10.3390/cryptography7030040.

[23] S. M. Hosseini and H. Pilaram, "A comprehensive review of post-quantum cryptography: Challenges and advances," *Cryptology ePrint Archive*, 2024, Paper 2024/1940. [Online]. Available: https://eprint.iacr.org/2024/1940.

[24] C. Sorzano, J. Vargas, and A. Pascual Montano, "A survey of dimensionality reduction techniques," *preprint arXiv:1403.2877*, 2014. [Online]. Available: https://arxiv.org/abs/1403.2877.

[25] M. Ashraf *et al.*, "A survey on dimensionality reduction techniques for time-series data," *IEEE Access*, vol. 11, pp. 42 909–42 923, 2023. doi: 10.1109/ACCESS.2023.3269693.

[26] Y. Wang, Y. Wang, and L. Wang, "Deep metric learning for few-shot image classification: A review of recent developments," *Pattern Recognition*, vol. 138, p. 109 381, 2023.

[27] J. Lu, J. Hu, and J. Zhou, "Deep metric learning for visual understanding: An overview of recent advances," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 76–84, 2017.

[28] Y. Luo, Y. Wen, L.-Y. Duan, and D. Tao, "Transfer metric learning: Algorithms, applications and outlooks," *Vicinagearth*, vol. 1, no. Article 2, 2024. doi: 10.1007/s44336-024-00003-8.

[29] M. T. Islam and L. Xing, "A data-driven dimensionality-reduction algorithm for the exploration of patterns in biomedical data," *Nature Biomedical Engineering*, vol. 5, no. 6, pp. 624–635, 2021. doi: 10.1038/s41551-020-00635-3.

[30] L. Fernández, M. Pérez, J. M. Orduña, and R. Martínez, "A new dimensionality reduction technique based on the wavelet transform for cancer classification," *Journal of Big Data*, vol. 12, p. 9, 2025. doi: 10.1186/s40537-024-01039-9.

[31] R. d. A. Coelho and N. S. D. Brito, "Analysis of rms measurements based on the wavelet transform," *Journal of Control, Automation and Electrical Systems*, vol. 32, no. 6, pp. 1588–1602, 2021. doi: 10.1007/s40313-021-00770-5.

[32] J. Lee, M.-H. Min, T.-P. An, and B.-G. Lee, "Comparison of computation performance for root mean square adapted to analyze fault data of railway power system," *Journal of Electrical Engineering & Technology*, vol. 15, no. 6, pp. 2841–2846, 2020. doi: 10.1007/s42835-020-00549-9.

[33] S. Yang and Q. Zhao, "Real-time frequency estimation for sinusoidal signals with application to robust fault detection," *International Journal of Adaptive Control and Signal Processing*, vol. 27, no. 5, pp. 386–399, 2013. doi: 10.1002/acs.2308.

[34] H. M. T. Menegaz, J. Y. Ishihara, G. A. Borges, and A. N. Vargas, "A systematization of the unscented kalman filter theory," *IEEE Transactions on Automatic Control*, vol. 60, no. 10, pp. 2583–2598, 2015.

[35] G. Younis, "Practical method to solve large least squares problems using cholesky decomposition," *Geodesy and Cartography*, vol. 41, no. 3, pp. 113–118, 2015. doi: 10.3846/20296991.2015.1086118.

[36] S. E. Mishchenko and N. V. Shatskiy, "An algorithm of angular superresolution using the cholesky decomposition and its implementation based on parallel computing technology," *Automatic Control and Computer Sciences*, vol. 57, pp. 661–671, 2023. doi: 10.3103/S014641162307009X.

[37] N. Hasan and L. Tiede, "Real-time global illumination with ray tracing and photon mapping," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 12, pp. 3873–3885, 2021.

[38] E. Gröller and D. Sgarbossa, "Ray tracing and mesh simplification for efficient rendering of large-scale 3d models," *Computer-Aided Design*, vol. 124, p. 102 795, 2020.

[39] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin, "Fastnerf: High-fidelity neural rendering at 200fps," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 2076–2085. doi: 10.1109/CVPR46437.2021.00208.

[40] M. Schütz, B. Kerbl, and M. Wimmer, "Rendering point clouds with compute shaders and vertex order optimization," *Computer Graphics Forum*, vol. 40, no. 2, pp. 43–53, 2021. doi: 10.1111/cgf.14130.

[41] I. Czerninski and Y. Y. Schechner, "Accelerating inverse rendering by using a gpu and reuse of light paths," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 10, pp. 4149–4160, 2021. doi: 10.1109/TVCG.2021.3064474.

[42] S. J. Byun, J. S. Hyun, and W. J. Sung, "Estimation of stochastic volatility and option prices," *Journal of Futures Markets*, vol. 41, no. 3, pp. 349–360, 2021. doi: 10.1002/fut.22168.

[43] Y. Jerbi and R. Bouzid, "Barrier options pricing under stochastic volatility using monte carlo simulation," *International Journal of Finance & Banking Studies*, vol. 12, no. 3, pp. 32–39, 2023. doi: 10.20525/ijfbs.v12i3.2851.

[44] H. Gong, A. Thavaneswaran, and J. Singh, "Stochastic volatility models with application in option pricing," *Journal of Statistical Theory and Practice*, vol. 4, no. 4, pp. 541–557, 2011. doi: 10.1080/15598608.2010.10412003.

preprint

Appendix A. C&P Notation

Table 4. *Chaos&Precision* Nomenclature for Square Roots ($\sqrt{n}$) where $1 \leq n \leq 100$

$\sqrt{1} = 1$	$\sqrt{2} = 1\left\{\frac{1}{2*}\right\}$	$\sqrt{3} = 1\left\{\frac{2}{2*}\right\}$	$\sqrt{4} = 2$	$\sqrt{5} = 2\left\{\frac{1}{4*}\right\}$
$\sqrt{6} = 2\left\{\frac{2}{4*}\right\}$	$\sqrt{7} = 2\left\{\frac{3}{4*}\right\}$	$\sqrt{8} = 2\left\{\frac{4}{4*}\right\}$	$\sqrt{9} = 3$	$\sqrt{10} = 3\left\{\frac{1}{6*}\right\}$
$\sqrt{11} = 3\left\{\frac{2}{6*}\right\}$	$\sqrt{12} = 3\left\{\frac{2}{6*}\right\}$	$\sqrt{13} = 3\left\{\frac{4}{6*}\right\}$	$\sqrt{14} = 3\left\{\frac{5}{6*}\right\}$	$\sqrt{15} = 3\left\{\frac{6}{6*}\right\}$
$\sqrt{16} = 4$	$\sqrt{17} = 4\left\{\frac{1}{8*}\right\}$	$\sqrt{18} = 4\left\{\frac{2}{8*}\right\}$	$\sqrt{19} = 4\left\{\frac{3}{8*}\right\}$	$\sqrt{20} = 4\left\{\frac{4}{8*}\right\}$
$\sqrt{21} = 4\left\{\frac{5}{8*}\right\}$	$\sqrt{22} = 4\left\{\frac{6}{8*}\right\}$	$\sqrt{23} = 4\left\{\frac{7}{8*}\right\}$	$\sqrt{24} = 4\left\{\frac{8}{8*}\right\}$	$\sqrt{25} = 5$
$\sqrt{26} = 5\left\{\frac{1}{10*}\right\}$	$\sqrt{27} = 5\left\{\frac{2}{10*}\right\}$	$\sqrt{28} = 5\left\{\frac{3}{10*}\right\}$	$\sqrt{29} = 5\left\{\frac{4}{10*}\right\}$	$\sqrt{30} = 5\left\{\frac{5}{10*}\right\}$
$\sqrt{31} = 5\left\{\frac{6}{10*}\right\}$	$\sqrt{32} = 5\left\{\frac{7}{10*}\right\}$	$\sqrt{33} = 5\left\{\frac{8}{10*}\right\}$	$\sqrt{34} = 5\left\{\frac{9}{10*}\right\}$	$\sqrt{35} = 5\left\{\frac{10}{10*}\right\}$
$\sqrt{36} = 6$	$\sqrt{37} = 6\left\{\frac{1}{12*}\right\}$	$\sqrt{38} = 6\left\{\frac{2}{12*}\right\}$	$\sqrt{39} = 6\left\{\frac{3}{12*}\right\}$	$\sqrt{40} = 6\left\{\frac{4}{12*}\right\}$
$\sqrt{41} = 6\left\{\frac{5}{12*}\right\}$	$\sqrt{42} = 6\left\{\frac{6}{12*}\right\}$	$\sqrt{43} = 6\left\{\frac{7}{12*}\right\}$	$\sqrt{44} = 6\left\{\frac{8}{12*}\right\}$	$\sqrt{45} = 6\left\{\frac{9}{12*}\right\}$
$\sqrt{46} = 6\left\{\frac{10}{12*}\right\}$	$\sqrt{47} = 6\left\{\frac{11}{12*}\right\}$	$\sqrt{48} = 6\left\{\frac{12}{12*}\right\}$	$\sqrt{49} = 7$	$\sqrt{50} = 7\left\{\frac{1}{14*}\right\}$
$\sqrt{51} = 7\left\{\frac{2}{14*}\right\}$	$\sqrt{52} = 7\left\{\frac{3}{14*}\right\}$	$\sqrt{53} = 7\left\{\frac{4}{14*}\right\}$	$\sqrt{54} = 7\left\{\frac{5}{14*}\right\}$	$\sqrt{55} = 7\left\{\frac{6}{14*}\right\}$
$\sqrt{56} = 7\left\{\frac{7}{14*}\right\}$	$\sqrt{57} = 7\left\{\frac{8}{14*}\right\}$	$\sqrt{58} = 7\left\{\frac{9}{14*}\right\}$	$\sqrt{59} = 7\left\{\frac{10}{14*}\right\}$	$\sqrt{60} = 7\left\{\frac{11}{14*}\right\}$
$\sqrt{61} = 7\left\{\frac{12}{14*}\right\}$	$\sqrt{62} = 7\left\{\frac{13}{14*}\right\}$	$\sqrt{63} = 7\left\{\frac{14}{14*}\right\}$	$\sqrt{64} = 8$	$\sqrt{65} = 8\left\{\frac{1}{16*}\right\}$
$\sqrt{66} = 8\left\{\frac{2}{16*}\right\}$	$\sqrt{67} = 8\left\{\frac{3}{16*}\right\}$	$\sqrt{68} = 8\left\{\frac{4}{16*}\right\}$	$\sqrt{69} = 8\left\{\frac{5}{16*}\right\}$	$\sqrt{70} = 8\left\{\frac{6}{16*}\right\}$
$\sqrt{71} = 8\left\{\frac{7}{16*}\right\}$	$\sqrt{72} = 8\left\{\frac{8}{16*}\right\}$	$\sqrt{73} = 8\left\{\frac{9}{16*}\right\}$	$\sqrt{74} = 8\left\{\frac{10}{16*}\right\}$	$\sqrt{75} = 8\left\{\frac{11}{16*}\right\}$
$\sqrt{76} = 8\left\{\frac{12}{16*}\right\}$	$\sqrt{77} = 8\left\{\frac{13}{16*}\right\}$	$\sqrt{78} = 8\left\{\frac{14}{16*}\right\}$	$\sqrt{79} = 8\left\{\frac{15}{16*}\right\}$	$\sqrt{80} = 8\left\{\frac{16}{16*}\right\}$
$\sqrt{81} = 9$	$\sqrt{82} = 9\left\{\frac{1}{18*}\right\}$	$\sqrt{83} = 9\left\{\frac{2}{18*}\right\}$	$\sqrt{84} = 9\left\{\frac{3}{18*}\right\}$	$\sqrt{85} = 9\left\{\frac{4}{18*}\right\}$
$\sqrt{86} = 9\left\{\frac{5}{18*}\right\}$	$\sqrt{87} = 9\left\{\frac{6}{18*}\right\}$	$\sqrt{88} = 9\left\{\frac{7}{18*}\right\}$	$\sqrt{89} = 9\left\{\frac{8}{18*}\right\}$	$\sqrt{90} = 9\left\{\frac{9}{18*}\right\}$
$\sqrt{91} = 9\left\{\frac{10}{18*}\right\}$	$\sqrt{92} = 9\left\{\frac{11}{18*}\right\}$	$\sqrt{93} = 9\left\{\frac{12}{18*}\right\}$	$\sqrt{94} = 9\left\{\frac{13}{18*}\right\}$	$\sqrt{95} = 9\left\{\frac{14}{18*}\right\}$
$\sqrt{96} = 9\left\{\frac{15}{18*}\right\}$	$\sqrt{97} = 9\left\{\frac{16}{18*}\right\}$	$\sqrt{98} = 9\left\{\frac{17}{18*}\right\}$	$\sqrt{99} = 9\left\{\frac{18}{18*}\right\}$	$\sqrt{100} = 10$

Proof

Appendix B. Pseudocodes (a) Newton-Raphson (b) Blended Trisection

function NewtonRaphsonSqrt($N, p, \text{max_iter} = 1000$)
  if $N = 0$ **then** ▷ Edge case
    return 0
  end if
  $x \leftarrow N/2$ ▷ Key initialization
  $\epsilon \leftarrow 10^{-p}$ ▷ Key initialization
  iter $\leftarrow 0$
  while $|x^2 - N| > \epsilon$ **and** iter $<$ max_iter **do** ▷ Threshold
    $x \leftarrow (x + N/x)/2$ ▷ Critical Operation
    iter $\leftarrow$ iter $+ 1$
  end while
  return x ▷ Return
end function

function Blended(N, $precision$)
  eps $\leftarrow 10^{-precision}$
  $a \leftarrow 0.0$
  $b \leftarrow N$
  if $N < 1$ **then**
    $b \leftarrow 1.0$
  end if ▷ Edge case
  $n \leftarrow 0$
  $a_1 \leftarrow a$, $a_2 \leftarrow a$ ▷ Key initialization
  $b_1 \leftarrow b$, $b_2 \leftarrow b$ ▷ Key initialization
  $f_a \leftarrow a^2 - N$ ▷ Key initialization

  repeat
    $n \leftarrow n + 1$
    ▷ Candidate points
    $x_{T1} \leftarrow (b + 2a)/3$ ▷ Trisection point 1
    $x_{T2} \leftarrow (2b + a)/3$ ▷ Trisection point 2
    $x_F \leftarrow a - \dfrac{f_a \cdot (b - a)}{f(b) - f_a}$ ▷ False position point
    ▷ Best candidate selection
    $x \leftarrow x_{T1}$
    $f_x \leftarrow f(x_{T1})$
    if $|f(x_{T2})| < |f_x|$ **then**
      $x \leftarrow x_{T2}$ ▷ Variable update
      $f_x \leftarrow f(x_{T2})$ ▷ Variable update
    end if
    if $|f(x_F)| < |f_x|$ **then**
      $x \leftarrow x_F$ ▷ Variable update
      $f_x \leftarrow f(x_F)$ ▷ Variable update
    end if
    ▷ Intervals update
    if $f_a \cdot f(x_{T1}) < 0$ **then**
      $b_1 \leftarrow x_{T1}$ ▷ Variable update
    else if $f(x_{T1}) \cdot f(x_{T2}) < 0$ **then**
      $a_1 \leftarrow x_{T1}$ ▷ Variable update
      $b_1 \leftarrow x_{T2}$ ▷ Variable update
    else
      $a_1 \leftarrow x_{T2}$ ▷ Variable update
    end if

    if $f_a \cdot f(x_F) < 0$ **then**
      $b_2 \leftarrow x_F$ ▷ Variable update
    else
      $a_2 \leftarrow x_F$ ▷ Variable update
    end if

    $a \leftarrow \max(a_1, a_2)$ ▷ Eval function
    $b \leftarrow \min(b_1, b_2)$ ▷ Eval function
  until $|f_x| \leq$ eps ▷ Threshold

  return x ▷ Return
end function

function f(x, N)
  return $x^2 - N$ ▷ Eval function
end function

preprint